Assembling Agent Oriented Software Engineering Methodologies from Features

Thomas Juan Leon Sterling Michael Winikoff

Department of Computer Science and Software Engineering, The University of Melbourne,
221 Bouverie Street Carlton, Victoria, 3010, Australia
`{tlj, leon}@cs.mu.oz.au`

School of Computer Science and Information Technology, RMIT University,
GPO Box 2476V, Melbourne, 3001, Australia
`winikoff@cs.rmit.edu.au`

Abstract. In this paper we describe our effort to merge two existing AOSE methodologies, Prometheus and ROADMAP, by isolating a set of general-purpose common elements. The remaining parts of the two methodologies are componentized into special purpose "value-adding" features. This approach empowers the developer to assemble a methodology tailored to the given project (the application domain) by adding appropriate features to the common elements. The assembled methodology can be modified during development to support changing aspects of the system.

1. Introduction

Many diverse Agent Oriented Software Engineering (AOSE) approaches and methodologies have been proposed [4, 14], including the AAII methodology, AUML, Gaia, MaSE, Tropos, Prometheus and ROADMAP [6, 8, 15, 3, 11, 10, 5]. Each of the methodologies has different strengths and weaknesses, and different specialized features to support different aspects of their intended application domains.

Clearly no single methodology is "one size fits all". However, as application complexity grows, we expect future projects to have an increasingly large number of aspects that cannot be addressed by a single methodology alone. To provide engineering support for such projects, specialized features to address different aspects must be brought together from different methodologies in a consistent fashion.

It is useful to identify and standardize the common elements of the existing methodologies. The common elements could form a generic agent model on which specialized features might be based. The remaining parts of the methodologies would represent "added-value" that the methodologies bring to the common elements, and should be "componentized" into modular features. The small granularity of features allows them to be combined into the common models in a flexible manner. By conforming to the generic agent model in the common elements, we expect the semantics of the optional features to remain consistent.

As a step towards standardizing methodologies, we investigate two in detail. Prometheus and ROADMAP [10, 5] have rather different aims. Prometheus focuses on building agents using BDI platforms [12] and on providing explicit and detailed processes and deliverables suitable for use by industry practitioners with limited agent experience, or by undergraduates. ROADMAP focuses on building open systems [7, 13, 16] and emphasis the societal aspects of an agent system. Key concepts in ROADMAP are roles and environmental zones. These concepts are used to model trust in open organizations of agents.

We introduce the two methodologies and discuss their respective applicability in section 2 and 3. We then identify a common core of models and processes in section 4 and show how both ROADMAP and Prometheus can be seen as adding to this common core in section 5 and 6. An illustrating example is presented in Section 7. Section 8 outlines future work and concludes.

2. Prometheus

The Prometheus[1] methodology is a detailed and complete ("start-to-end") process for specifying, designing, and implementing intelligent agent systems, which has been developed over the last few years in collaboration with Agent Oriented Software[2] (A company which markets the agent development software platform JACK™ [2], as well as agent solutions). The goal in developing Prometheus is to have a process with defined deliverables, which can be taught to industry practitioners and undergraduate students who do not have a background in agents, and which they can use to develop intelligent agent systems.

The methodology consists of three phases: system specification, architectural design, and detailed design. The following description of the phases and deliverables is necessarily brief, for more information see [10].

System Specification: Actions and percepts define the interface between agents and the environment in which they are situated. Functionalities describe in a broader sense what the system should do (e.g. *"the robot will try to maintain an awareness of ball position"*). In this phase actions, percepts and functionalities are identified and specified. Use case scenarios are created to provide a more holistic view of the interaction between actions, percepts and functionalities.

Architectural Design: The major decision to be made during the architectural design is which agents should exist within the system, and what functionalities they should have. The artifacts of the previous phase are analyzed to suggest possible designs, which are evaluated according to the traditional software engineering criteria of coherence and coupling.

During this stage of the design it is important to identify the events (significant occurrences) that the agent will respond to. Agent messages are also identified, forming the interface between agents. At this point interaction protocols are specified

---

[1] Prometheus was the wisest Titan. His name means "forethought" and he was able to foretell the future. Prometheus is known as the protector and benefactor of man. He gave mankind a number of gifts including fire. (*http://www.greekmythology.com/*)

[2] http://www.agent-software.com

based on interaction diagrams (which in turn are based on use cases). If there are to be any shared data objects in the system, these should also be identified at this stage.

The system overview diagram ties together the agents, events and shared data objects. It is arguably the single most important artifact of the entire design process, although it cannot be understood fully in isolation.

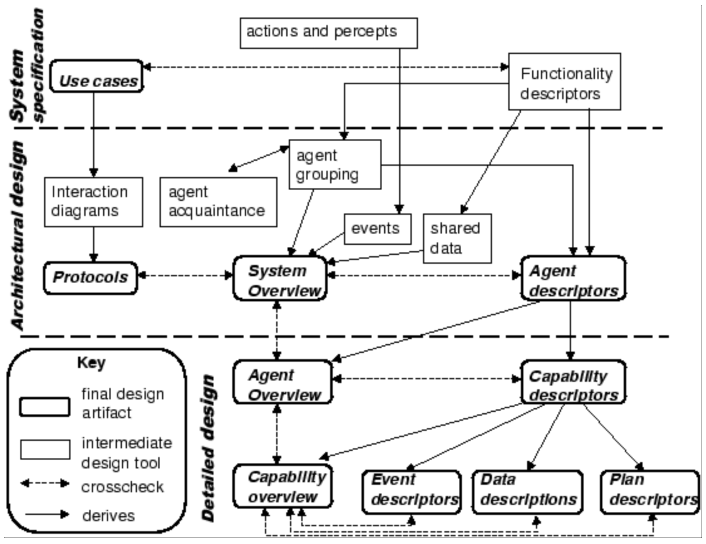

Fig. 1. Overview of 3 Phases in Prometheus

Detailed Design: Detailed design focuses on developing the internal structure of each of the agents and how it will achieve its tasks within the system. The focus is on defining capabilities (modules within the agent), in terms of internal events, plans and detailed data structures.

Because design in Prometheus is oriented towards systems of *intelligent* agents, each agent may well be quite complex, with a number of interacting responsibilities, and an ability to fulfil these responsibilities in a variety of ways. Capabilities can be nested which allows for arbitrarily many layers within the detailed design, in order to achieve an understandable complexity at each level.

Applicability: Prometheus supports the engineering of conventional closed systems with controlled and trusted agents. It specifically supports the BDI framework, and focuses on functionalities. Its concrete nature and detailed models and processes allow easy transition from the conventional OOSE approaches and make it very suitable for conventional applications such as an intelligent web server.

However, it lacks support for advanced properties such as openness and is not suitable for systems requiring these properties.

3. ROADMAP

The ROADMAP[3] methodology aims to support the engineering of large-scale open systems. It extends the Gaia methodology [15] by introducing use-cases for requirement gathering, explicit models of agent environment and knowledge, and a interaction model based on AUML interaction diagrams [8].

The original Gaia role model is also extended with a dynamic role hierarchy. This role hierarchy is carried into design and will have a run-time realization, allowing social aspects to be explicitly modeled, reasoned and modified at run-time.

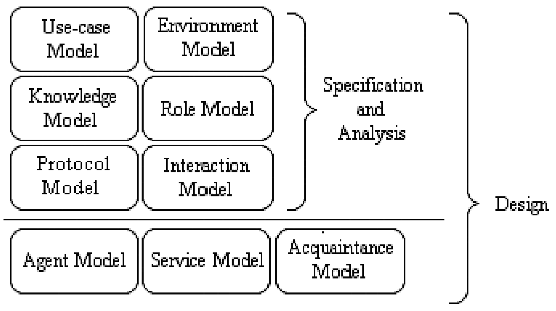

Fig. 2. Overview of ROADMAP

ROADMAP promotes the view of software systems as computational organizations. Agents in a system are similar to individuals in a human organization, while the roles in ROADMAP encapsulate regulations, processes, responsibilities and team roles by which individuals function within a human organization. They specify, support and constraint an agent's behaviors in the organization. When the expected behaviors in the organization is explicitly represented at run-time, agents can verify each other's behavior, and misbehaving agents can be identified and removed or replaced.

A useful level of trust can be established when new agents enter the system. If the new agent has the appropriate knowledge, and behaves according to its role in the correct environment zone, then the other agents in the organization can trust this agent to act to achieve the overall goal of the organization.

The relationship between roles and agents is similar to the relationship between interfaces and objects in OO approach. Like interfaces, roles provide an abstract model of the system above concrete implementation of functionalities. However, unlike interface, roles can be changed at run-time given the correct authorization. Instead of an immutable contract of behavior, roles should be considered as a long-term agreement of behavior that can be reasoned and changed. This difference allows a computing organization modeled in roles to be more flexible.

For more information on ROADMAP, please see [5].

[3] This is the acronym for Role-Oriented Analysis and Design for Multi-Agent Programming

The Development Process: The ROADMAP methodology encourages an iterative approach and expects details of the models to be filled in when applicable. During the analysis phase, the six analysis models are created to allow conceptualization of the system as an organization. During the design phase, the initial conceptualization of the system is optimized for the chosen quality goals, such as performance. The original models are modified and refined to reflect the design decisions. In addition, three new design models are created to populate the updated organization with member agents.

Applicability: Before adopting the feature-based approach, ROADMAP prescribes rich models to explicitly deal with roles of agents, the environment and knowledge in the system. The methodology provide strong support for engineering complex open systems, but is less suitable for application not requiring these properties. Consider a stand-alone desktop productivity tool where little knowledge is required outside its functionalities. Given its static nature, simple environment and lack of knowledge, creating the models prescribed by ROADMAP simply causes extra overhead.

4. The Common Elements

In this section we present a skeleton methodology, consisting of the common elements identified from Prometheus and ROADMAP. Each common element is then described in more detail. The analysis to derive the common elements is straightforward and omitted here.

Overview

The skeleton methodology is independent of the implementation architecture and supports analysis and architecture design of the system. It consists of six basic models as depicted in Figure 3. During the analysis stage, the models are created to conceptualize and document the system requirements. During the architecture design stage, the same models are refined and optimized for the given quality goals. During these two stages, agents are considered as black boxes. At the end of the architecture design phase, an implementation architecture, such as the BDI architecture, is chosen for each agent.

The skeleton methodology does not support detailed design, and relies on optional features from other architecture-dependent methodologies to fill the gap. In Figure 3, features derived from the Prometheus methodology are available to support the design of agent internals with BDI constructs.

The ROADMAP features are architecture-independent and can be added to the skeleton methodology to facilitate analysis and high-level design. Each feature can be introduced into a few models in the skeleton methodology, forming a thread related to and addresses the same issue.

The process of the skeleton methodology is simple. A developer can create the models in order of listing, and fill in details of any model when possible.

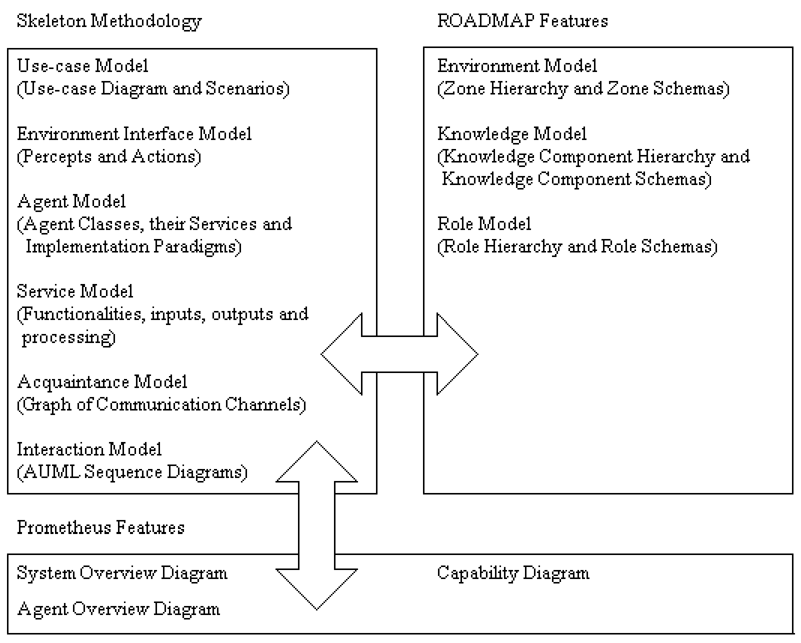
Skeleton Methodology
ROADMAP Features
Use-case Model
(Use-case Diagram and Scenarios)
Environment Interface Model
(Percepts and Actions)
Agent Model
(Agent Classes, their Services and Implementation Paradigms)
Service Model
(Functionalities, inputs, outputs and processing)
Acquaintance Model
(Graph of Communication Channels)
Interaction Model
(AUML Sequence Diagrams)
Environment Model
(Zone Hierarchy and Zone Schemas)
Knowledge Model
(Knowledge Component Hierarchy and Knowledge Component Schemas)
Role Model
(Role Hierarchy and Role Schemas)
Prometheus Features
System Overview Diagram
Agent Overview Diagram
Capability Diagram

Fig. 3. Overview of the skeleton methodology and optional features from Prometheus and ROADMAP

The Skeleton Models

The models in the skeleton methodologies are as listed:

1. The Use-case Model is adapted from the conventional OO use-case model [1, 9]. It contains graphical use-case diagrams and text scenarios. The main difference is that agents in the system are depicted. Figure 4 shows an example.
2. The Environment Interface Model is a list of percepts and actions possible in the environment of the agents.
3. The Agent Model lists agent classes, the services they provide, and the implementation paradigm chosen for them. This is similar to the class model in conventional OO approaches.
4. The Service Model contains basic descriptions, such as input, output and the processing, for each service in the system
5. The Acquaintance Model is a directed graph between agent types. An arc in the graph represents the existence of a communication link allowing messages to be sent. The purpose is to allow the designer to visualize the degree of coupling between agents. In this light, details such as message types and orders are ignored.
6. The Interaction Model contains sequence diagrams as proposed in AUML.

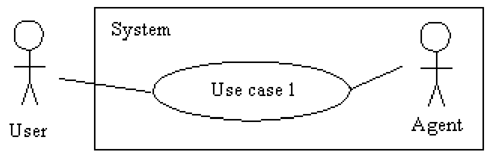

Fig. 4. An example use case diagram

Summary
The models contained in the skeleton methodology are designed to be simple and closely related to conventional OO models for two reasons. It allows developers without background in agent research to easily utilize the technology.

It also reduces the overhead from unused advanced features. For systems requiring low level of agency, the cost of development with the skeleton methodology is that similar to with OO methodologies. When required, a threaded approach ensures the skeleton models are consistently updated with a given feature. This enables a smooth transition to model different levels of intelligence and agency.

5. The Optional Features from ROADMAP

ROADMAP introduces three threads of related features to complement the skeleton methodology. They model environmental zones, domain knowledge and the roles in the system.

1. The environment model replaces the original environment interface model in the skeleton methodology. Instead of a uniform space, the environment can now be modeled in zones. The environment model contains a hierarchy of zones in the environment, and a set of zone schema to describe each zone in the hierarchy. The zone hierarchy is similar to an OO class hierarchy and uses inheritance and aggregation to relate zones and the objects in zones. Figure 5 provides an example.

   Percepts and actions are now modeled as read and write methods of objects in the zone controllable by the agents. Static objects model entities in the zone whose existence is known to the agents, but no direct interaction exists.

   Zones can be added to the use-case model, the agent model, the acquaintance model and the interaction model. Please see Figure 6 for an example use-case and Figure 7 for an example interaction diagram.

2. The knowledge model formalizes the knowledge aspect of the system. By abstracting knowledge out of functionalities, we aim to facilitate sharing and re-using of knowledge. It will also encourage the system to be designed and implemented at a higher level.

   A knowledge component is a coherent block of knowledge. The knowledge model consists of a knowledge component hierarchy, and a set of schema for each knowledge component. Knowledge components can be added into the use-case model, where required knowledge components to achieve the scenarios are identified. Knowledge components can also be assigned to roles, to model knowledge distribution in the organization.

3. The role model allows the developer to express an abstract framework of the organization according to which its member agents behave. The role model consists of a role hierarchy and a set of role schemas for each role in the hierarchy. The roles define expected behaviors of the agents and provide verification services to any authorized agents at run-time.

   The agent model can include a list of roles taken by each agent. Agents interacting through roles can also be modeled in the use-case model, the acquaintance model and the interaction model. Please see Figure 6 and 7 for examples of roles in use-case model and interaction model.

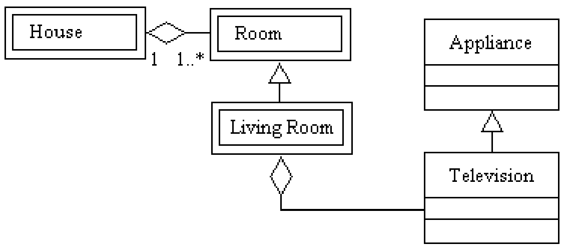

Fig. 5. An example of Zone Hierarchy with Objects

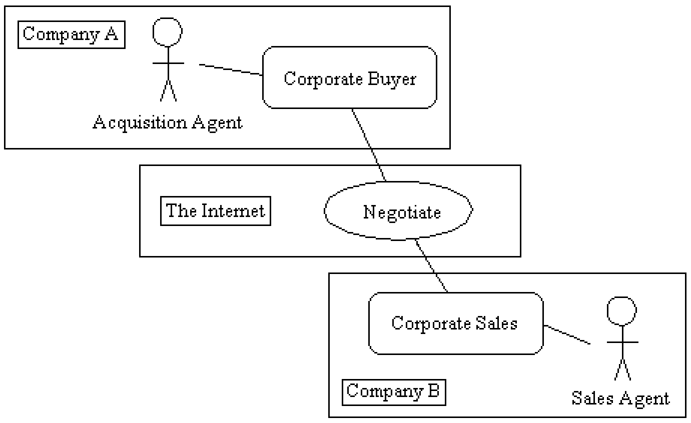

Fig. 6. An Example of Use-case diagram showing roles and zones

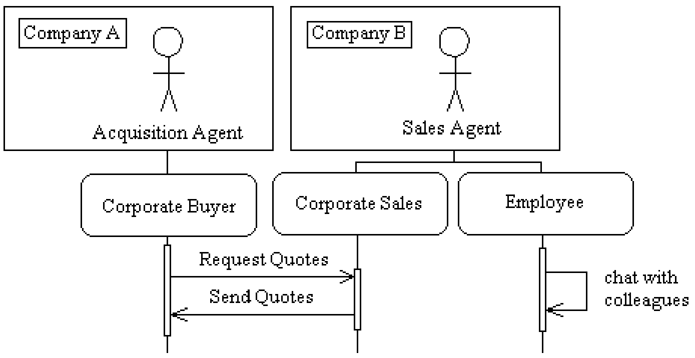

Fig. 7. An Example of Interaction Diagram with Roles and Zones

6. The Optional Features from Prometheus

Prometheus introduces one thread of features for detailed design using BDI constructs. The key concepts are beliefs, events, plans and capabilities as outlined in Section 2. The architecture-independent models from the skeleton methodology were largely based on the original Prometheus models and can be refined directly to the BDI oriented models.

1. System overview diagram shows agents in the system and the events they send to each other. This diagram can be derived directly from the skeleton use-case model and the interaction model. Figure 8 shows an example.

2. Agent overview diagram shows the internal working of an agent as interacting capabilities and beliefs through event passing. Figure 9 shows an example.

3. Capability diagram shows the internal working of a capability as interacting plans through event passing. Figure 10 shows an example.

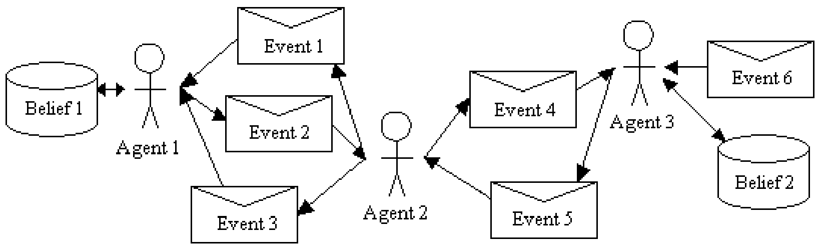

Fig. 8. Example System Overview Diagram

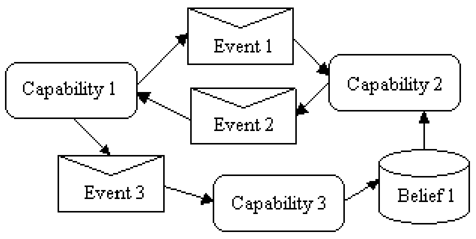

Fig. 9. Example Agent Overview Diagram

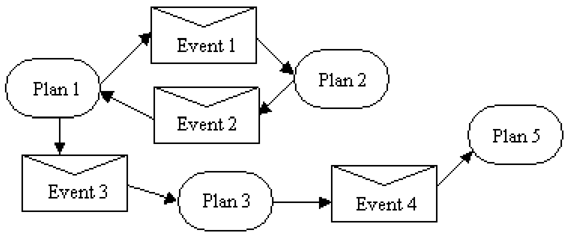

Fig. 10. Example Capability Overview Diagram

The Prometheus thread is straightforward. For each BDI construct, such as plans and events, descriptors can be created to express additional implementation information.

7. Illustrating Example: Extending an Intelligent Personal Assistant

We illustrate our proposed feature-based approach by a scenario. Consider a business attempting to boost employee productivity by providing an intelligent personal assistant. The assistant allows the users to communicate via instant messaging, e-mail, voice conversation and file sharing. It will also allow the users to publish and manage their timetables, and allow appointments to be made online.

Developers in the company are not familiar with the agent paradigm and used the skeleton methodology to model the functionalities. At the end of the architecture phase the BDI architecture is chosen for the implementation. The optional features derived from Prometheus are used to create the detailed design and translate easily into Jack codes.

The prototype was a success and the company decides to integrate more aspects of its operation into the Personal Assistant. To accommodate the richer requirements, the development team extends the skeleton methodology with optional features derived from ROADMAP.

The employees want access to the Personal Assistant at home, from their PDAs and mobile phones. They also request integration of the Personal Assistant to the infrastructures, ie, task tracking, of various projects and departments they are involved in. Suppliers and clients of the company also requested to be included in the network. Environment zones and related features are therefore added into the development methodology to model the security and access control of various access locations and modes. Roles are used to model employee involvement in projects and departments. By structuring the system as an agent organization, the system can be modified and extended easily.

The knowledge modeling thread is also introduced to abstract domain knowledge out of functionalities. Business rules in projects and departments can then be shared and re-used.

The original analysis and design models were consistently extended with the optional features to handle the new requirements. The threaded approach to build flexible methodologies significantly reduced the development risks in this example.

8. Conclusion and Future Work

We expect future software to require in addition to traditional quality goals such as performance, security or scalability, a new set of less precise quality goals such as privacy, politeness and good taste. These quality goals arise from our increasing reliance on software to decide and act on our behalves.

The engineering of these quality goals requires new methods. For example, we might need to analyze the distribution of knowledge in the system and answer questions like: does any agent have unnecessary knowledge that can be abused? What do we do to agents who know too much?

It's worth noting that in large-scale systems, the engineering of quality goals is mainly required on particular parts of the system. For example, politeness is important at the user interface. It's most cost effective to focus the engineering effort locally to the appropriate module, instead of the entire system.

The threaded approach allows new techniques to be deployed into projects consistently, and applied accurately to the part of the system requiring the effort. The skeleton models could model the functionalities of the entire system, and form a foundation for specialized features to be added. The optional feature would support the engineering of quality goals important to local modules. For example, features supporting the engineering of politeness only need to be applied to models concerning the user interface.

In this paper we described our experience in merging two existing methodologies, Prometheus and ROADMAP. We presented a useful approach to collect common elements into a skeleton methodology, and componentize the remainders of the methodologies into complementary threads of features. We described the actual generic models and features obtained from applying this approach to Prometheus and ROADMAP, and illustrate the usefulness of this approach with an example.

In future we aim to formalize and standardize the agent model underlying the merged methodologies. The standard agent model could allow other methodologies to be componentized and merged with consistent semantics.

References

1. Booch, G. Object-Oriented Analysis and Design (2nd edition). Addison-Wesley: Reading, MA, 1994.
2. Busetta, P., Ronnquist, R., Hodgson, A and Lucas, A, Jack Intelligent Agents – Components for Intelligent Agents in Java. Agent Oriented Software Pty. Ltd, Technical Report tr9901, 1999. http://www.agent-software.com
3. DeLoach, S., Analysis and Design using MaSe and agentTool, Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2002), 2001.
4. Iglesias, C., Garijo, M., and Gonzalez, J. A Survey of Agent-Oriented Methodologies. In Intelligent Agents V - Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg. 1999
5. Juan, T., Pearce, A. and Sterling, L., Extending the Gaia Methodology for Complex Open Systems, Proceedings of the 2002 Autonomous Agents and Multi-Agent Systems, Bologna, Italy, July 2002.
6. Kinny, D., Georgeff, M. and Rao, A., A Methodology and Modeling technique for systems of BDI agents. Proceedings of the 7th European workshop on modeling autonomous agents in a multi-agent world, LNCS 1038, p56-71, Springer-Verlag, Berlin Germany1996
7. Ladaga, R., Active Software, in Self-Adaptive Software, P. Robertson, H. Shrobe, and R. Lagada, Editors. 2000, Springer-Verlag: New York, NY. P.11-26
8. Odell, J., Parunak, H. and Bauer, B., Extending UML for agents. In the Proceedings of the Agent-Oriented Information System Workshop at the 17th National Conference on Artificial Intelligence, 2000.
9. OMG. OMG Unified Modeling Language Specification. 1999 http://www.rational.com/media/uml/post.pdf
10. Padgham, L. and Winikoff, M., Prometheus: A Methodology for Developing Intelligent Agents, Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS 2002. July, 2002, Bologna, Italy.
11. Perini, A., Bresciani, P., Giunchiglia, F., Giorgini, P and Mylopoulos, J., A knowledge level software engineering methodology for agent oriented programming. Proceedings of Autonomous Agents, Montreal CA, 2001.
12. Rao, A. and Georgeff, M, BDI-agents: from theory to practice, Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco, 1995.
13. Robertson, P., R. Ladaga, and H.Shrode, Introduction: The First International Workshop on Self-Adaptive Software, in Self-Adaptive Software, P.Robertson, H.Shrobe, and R. Ladaga, (eds). 2000, Springer-Verlag: New York, NY. P.11-26
14. Wooldridge, M. and Ciancarini, P. Agent-Oriented Software Engineering: The State of the Art. In Agent-Oriented Software Engineering. Ciancarini, P. and Wooldridge, M. (eds), Springer-Verlag Lecture Notes in AI Volume 1957, 2001.
15. Wooldridge, M., Jennings, N. and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3 (3). 2000, 285-312.
16. Yokote, Y. The Apertos Reflective Operating System: The Concept and its Implementation, Proc. OOPSLA `92, ACM, 1992, p414-434.